Research at the Scalable Computing Software Laboratory

Xian-He Sun

Zhiling Lan

Department of Computer Science
Illinois Institute of Technology
www.cs.iit.edu/~scs/

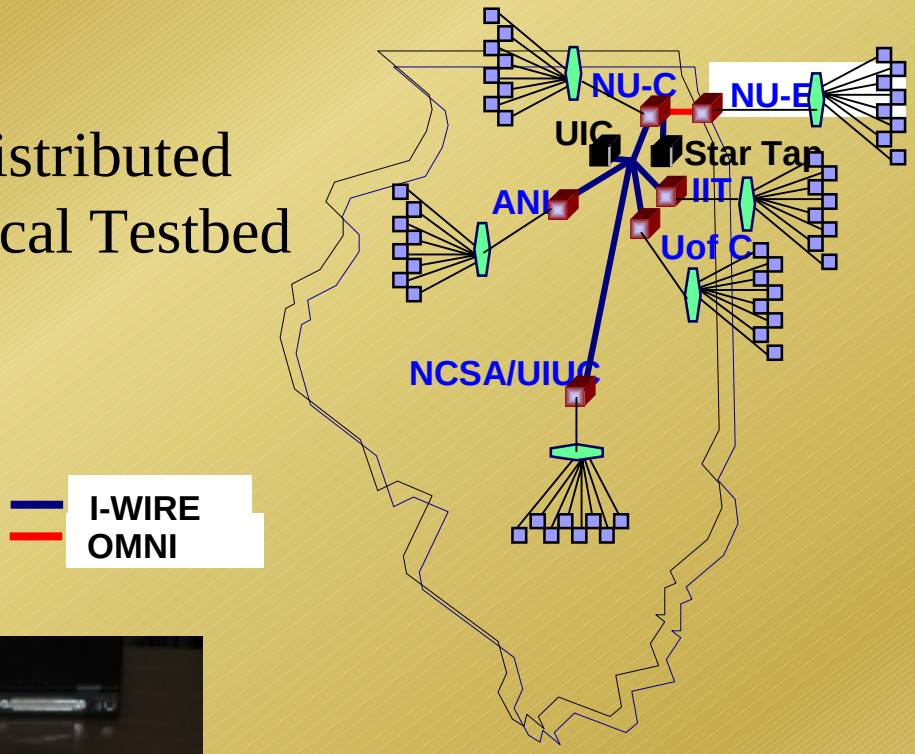
Fermi Lab Presentation

SCS

SCS Computing Infrastructure

Parallel Computers

Distributed
Optical Testbed



Pervasive Computing
Environments

X. Sun, Feb. 2004

SCS

Outline

- High Performance Computing
  - Scalable numerical kernel solvers
  - Performance optimization
- Distributed Computing
  - Mobility and mobility of legacy code
  - Performance prediction and task scheduling
- Pervasive Computing (not cover)
- Application
- Conclusion

SCS

Scalable Numerical Algorithms

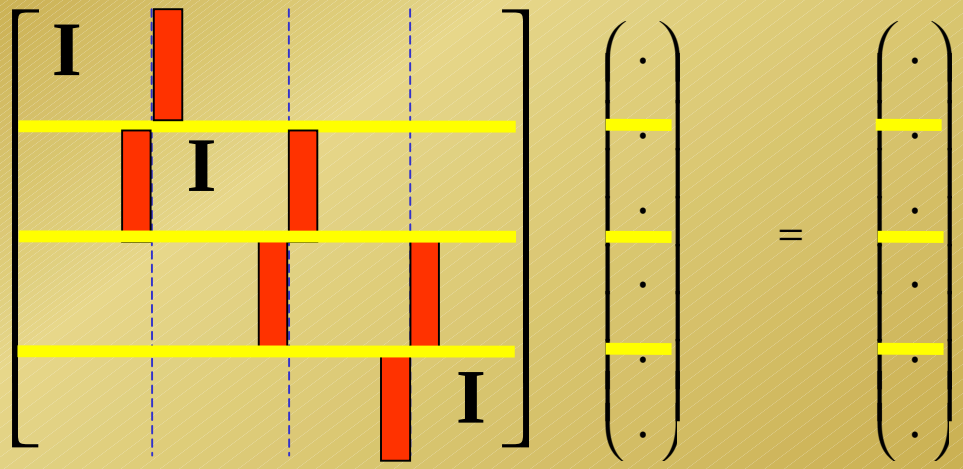
- Motivation
  - Parallel codes have been developed during last decade
  - The performances of many codes suffer in a scalable computing environment
- Achievement
  - Scalable tridiagonal solvers
  - Fast and high-order Poisson solvers
  - Iterative Helmholtz equation algorithms
  - Domain decomposition methods

SCS

Sample: The PDD algorithm & its applications

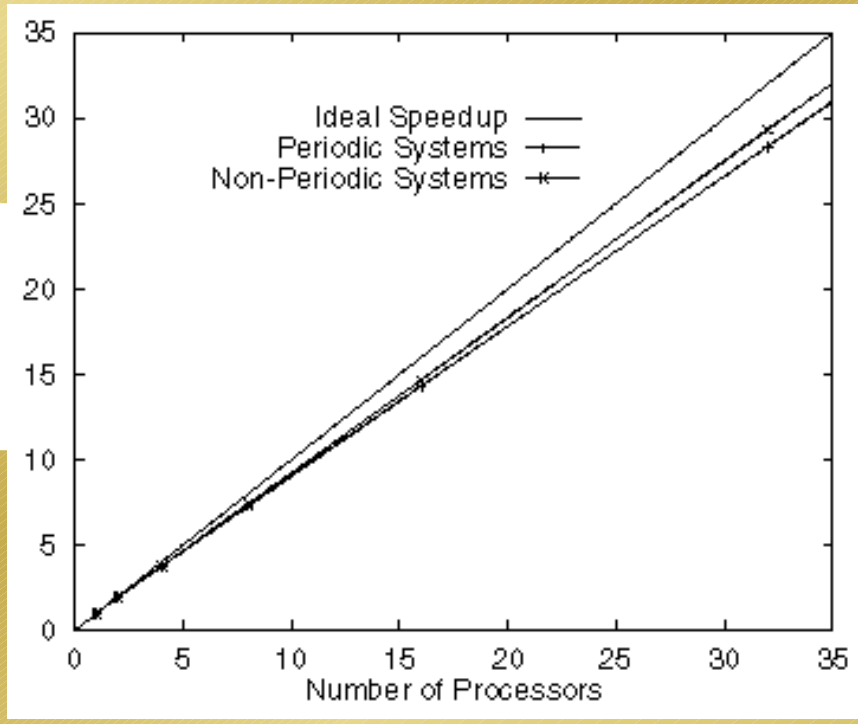
The Parallel Diagonal Dominant (PDD)

The Reduced System

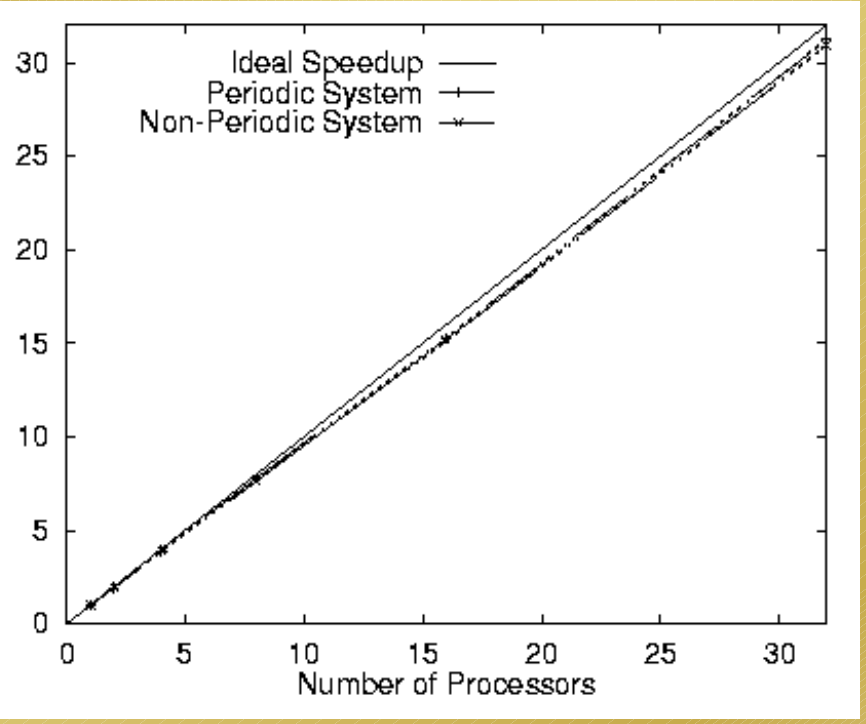


Generally needs global communication, Decay for diagonal dominant systems

SCS



Scaled Speedup of the PDD Algorithm on Paragon. *1024 System of order 1600, periodic & non-periodic*



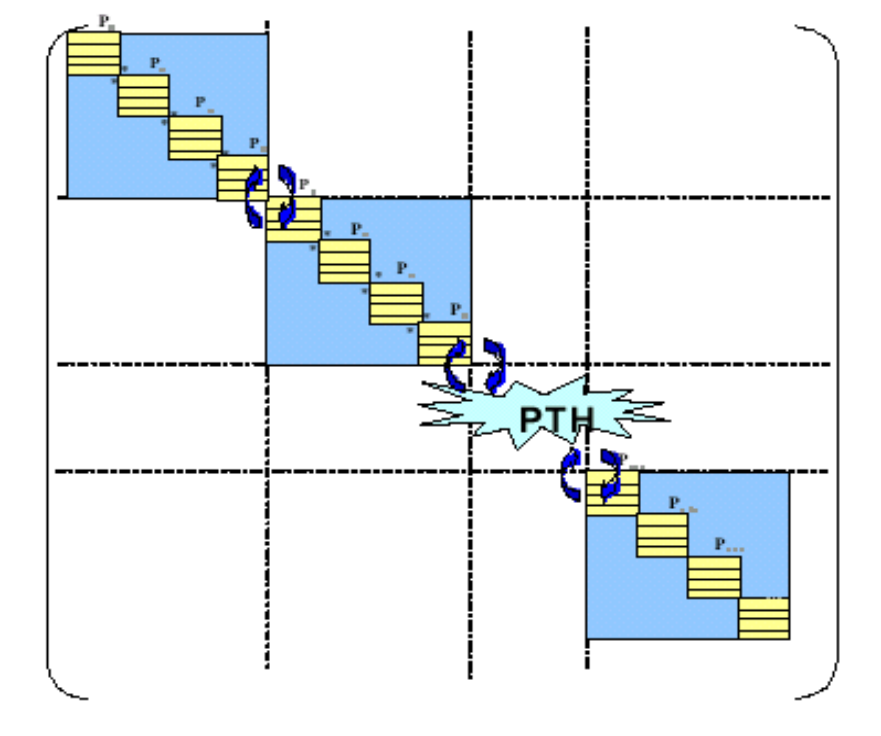
Scaled Speedup of the Reduced PDD Algorithm on SP2. *1024 System of Order 1600, periodic & non-periodic*

X. Sun, Feb. 2004

SCS

The Parallel Two-Level Hybrid Method

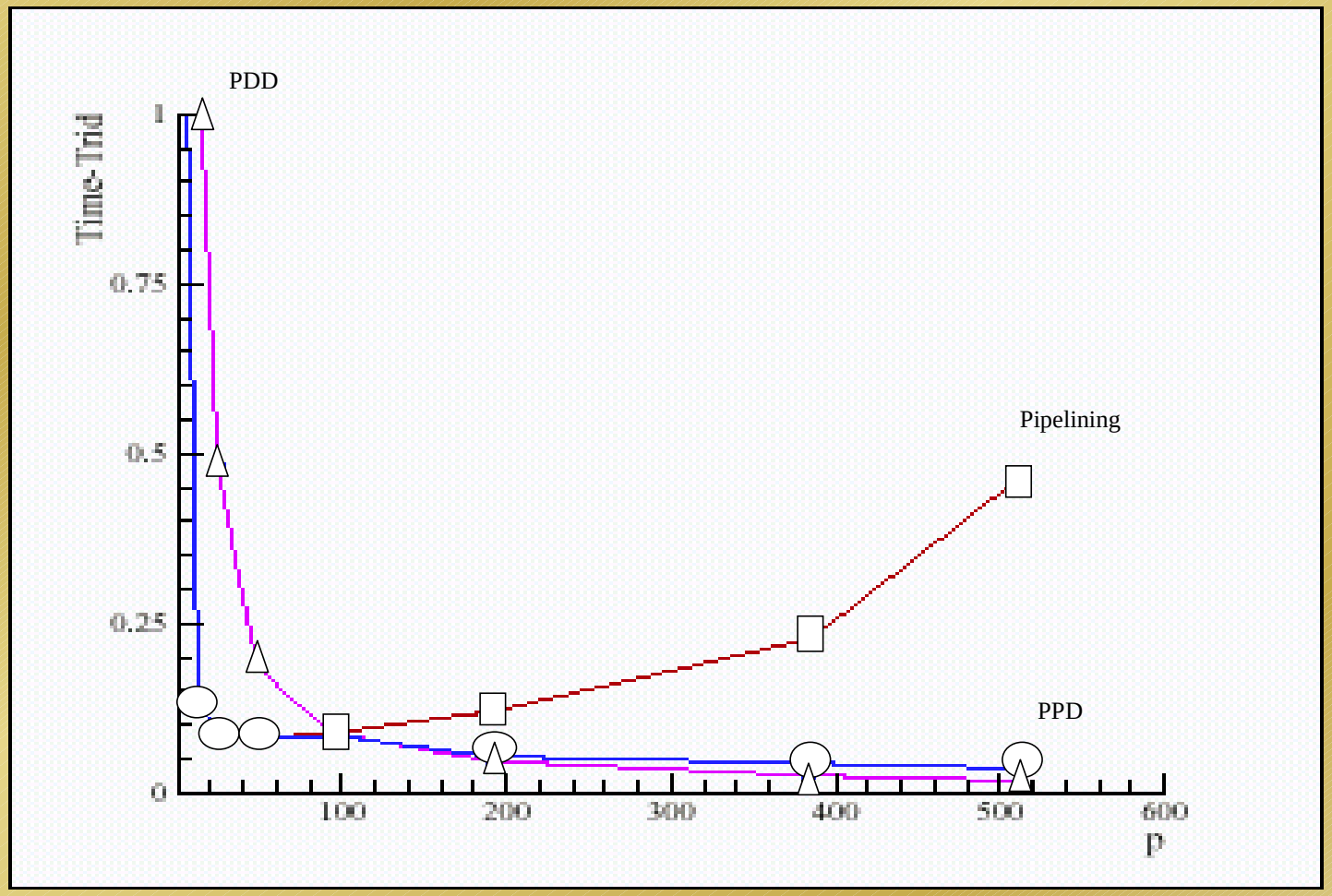
Combine the PDD with the Pipelining



- Use an accurate parallel tridiagonal solver to solve the *m* super-subsystems concurrently, each with k processors
- Modify PDD algorithm and consider communications only between the *m* super-subsystems.

SCS

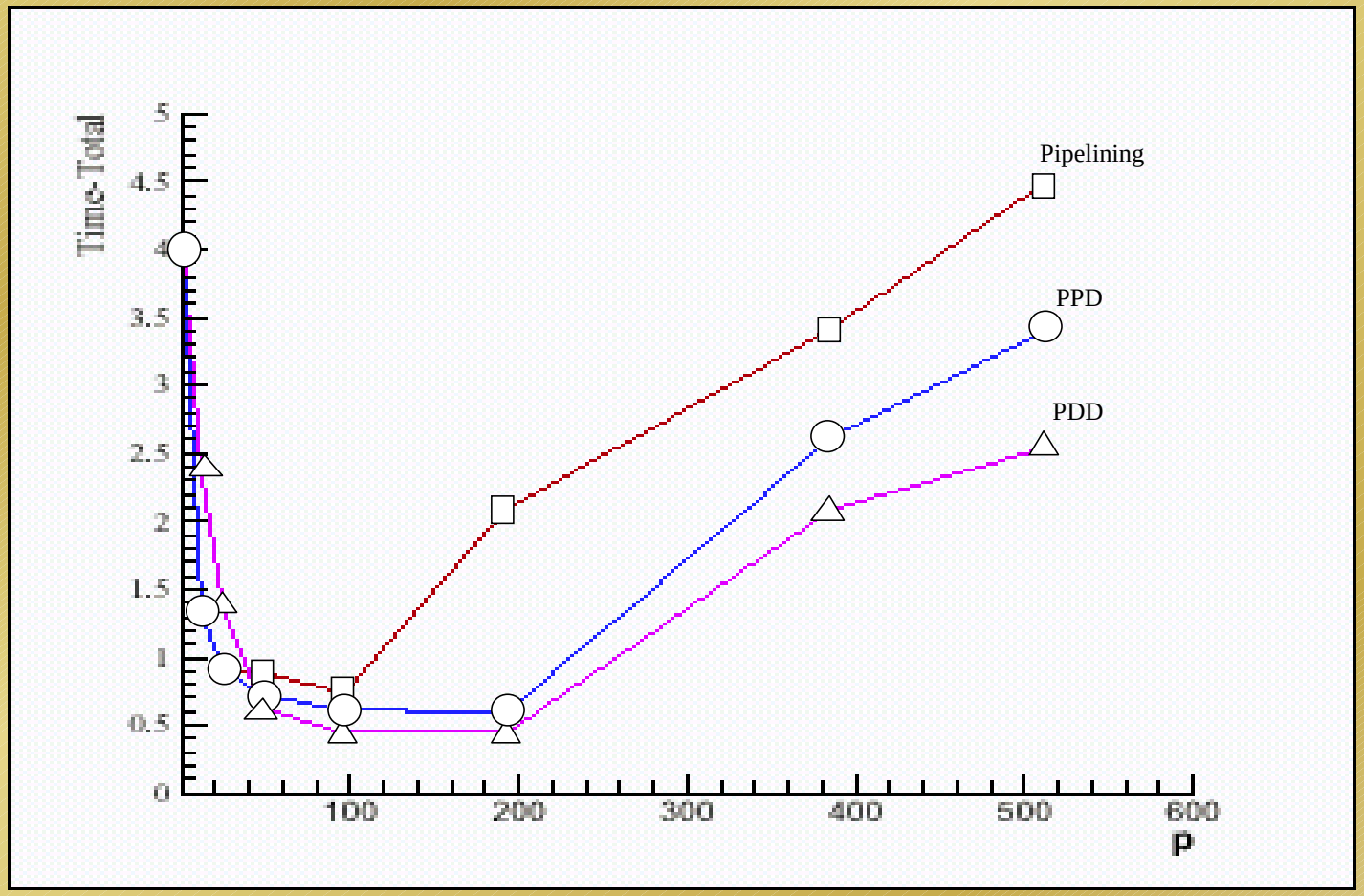
PDD, Pipelining, PPD (hybrid) for NLOM Tridiagonal Systems



X. Sun, Feb. 2004

SCS

PDD, Pipelining, PPD for NLOM Poisson Equations



X. Sun, Feb. 2004

SCS

Distributed Computing

High Performance Computing Mobility (HPCM)

- With the rapid advance of communication, the next generation computing will be: Mobile Computing
- Current successes of mobile computing are based on safe-languages such as Java, which is slow and cannot apply to legacy codes
- The HPCM middleware makes codes written in traditional languages such as Fortran, C, C++ migratable

SCS

Technical Challenges of HPC mobility

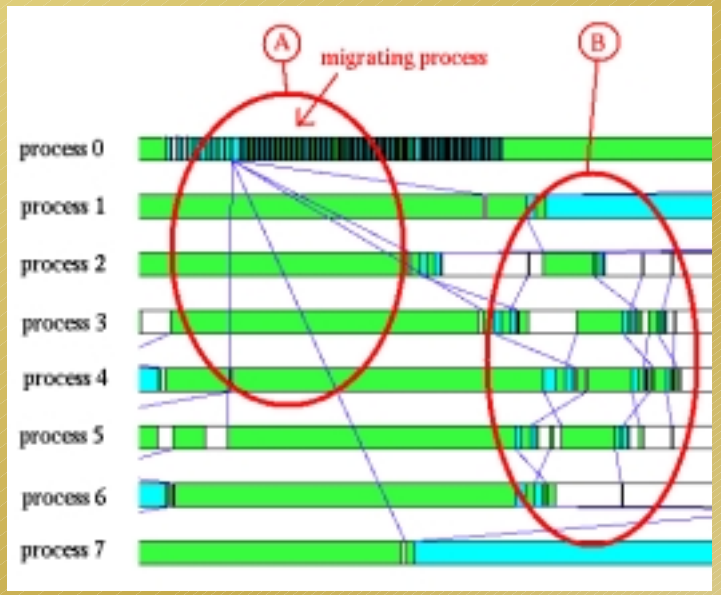
- How to transfer Execution State?
- How to transfer Memory State?
- How to transfer Communication State?
- How to transfer process state efficiently?
- How to transfer process state automatically?
- How to support process migrate from one virtual organization to another in a Grid environment?
- How to support mobility in hybrid Java-native code environment?
- How to design a coordinated middleware?

SCS

Mobility of Legacy Code

continue

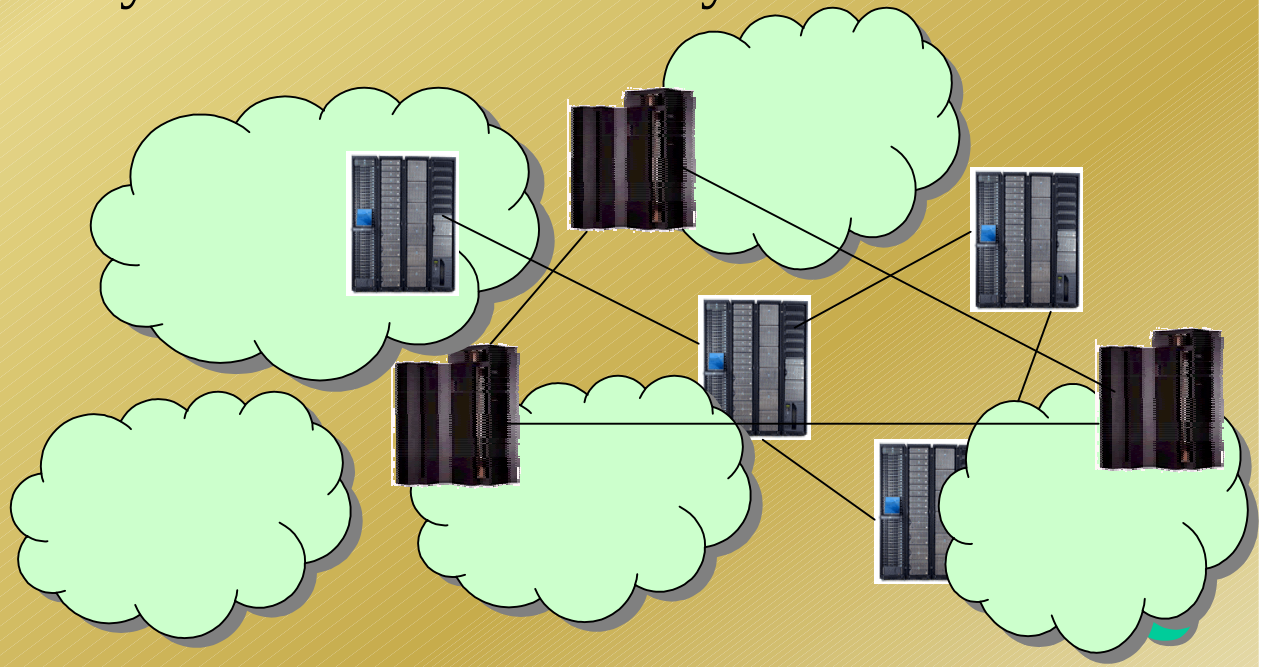
- We have developed novel methodologies and a prototype system, HPCM, to migrate codes written in traditional languages such as Fortran, C, C++
  - –Two level mobility: migrate native codes under Java virtual machine
  - –General methods: migrate between different computing systems and different virtual organizations.
  - –Leading technology, strong mobility



SCS

Performance and Scheduling

- New challenge in Grid Computing
  - Resources are shared
  - Data are shared resources
- New challenge in high performance computing
  - Memory hierarchy and data access delay



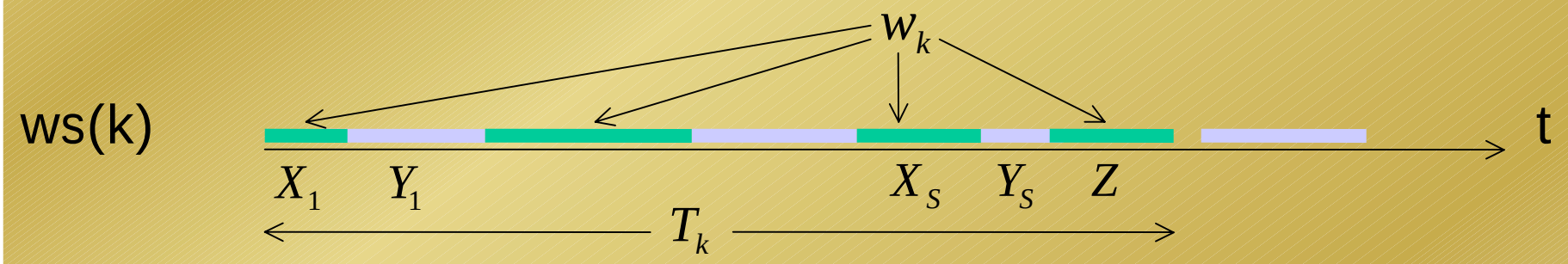
The Grid Harvest Service (GHS) System

- A long-term application-level performance prediction and scheduling system for non-dedicated distributed (Grid) environments
- A new prediction model derived via probability analysis and simulation
- New scheduling heuristics for resource, QoS, and data conscious scheduling
- Runtime dynamic scheduling

SCS

Performance Model

- Remote job has low priority
- Local job arriving and service time based on extensive monitoring and observation



$$T_k = X_1 + Y_1 + X_2 + Y_2 + \cdots + X_S + Y_S + Z$$

$$T_k = w + Y_1 + Y_2 + \cdots + Y_S$$

SCS

Prediction Formula

- Parallel task completion time

$$\Pr(T \le t) = \begin{cases} \prod_{k=1}^{m} [e^{-\lambda_k w_k} + (1 - e^{-\lambda_k w_k}) \Pr(U(S_k) \le t - w_k \mid S_k > 0)], & if\ .t \ge w_{\max} \\ 0, & otherwise \end{cases}$$

- Homogeneous parallel task completion time

$$\Pr(T \le t) = \begin{cases} [e^{-\lambda w} + (1 - e^{-\lambda w}) \Pr(U(S) \le \tau \mid S > 0)]^m, & if \quad \tau > 0 \\ 0, & otherwise \end{cases}$$

where, $\tau = t - w$

- Mean time balancing partition

$$w_k = \frac{W}{\sum_{k=1}^{m} (1 - \rho_k)\tau_k} (1 - \rho_k)\tau_k$$

X. Sun, Feb. 2004

SCS

Scheduling Algorithms

Scheduling with a Given Number of Sub-tasks

List a set of lightly loaded machines $M = \{m_1, m_{2,} \ldots m_q\}$;

List all possible sets of machines, such as $|S_i| = p$

For each machine set $S_k \quad (1 \le k \le z)$,

Use mean time balancing partition to partition the task

Use the formula to calculate the mean and coefficient of variation

If $E(T_{S_{p'}})(1 + Coe.(T_{S_{p'}})) > E(T_{S_k})(1 + Coe.(T_{S_k}))$, then $p' = k$;

End For

Assign parallel task to the machine set $S_{p'}$;

SCS

Optimal Scheduling Algorithm

List a set of lightly loaded machines $M = \{m_1, m_{2,} \ldots m_q\}$;

While $p < q$ **do**

Scheduling with p Sub-tasks

If $E(T_{S_{k'}{}^{p'}})(1+Coe.(T_{S_{k'}{}^{p'}})) > E(T_{S_k{}^{p}})(1+Coe.(T_{S_k{}^{p}}))$ then

$p' = p$;

End If

End while

Assign parallel task to the machine set $S_{k'}{}^{p'}$.

SCS

Heuristic Scheduling Algorithm

- List a set of lightly loaded machines $M = \{m_1, m_{2,} \ldots m_q\}$;
- Sort the machines in a decreasing order with $(1-\rho_k)\tau_k$;
- Use the task ratio to find the upper limit q ;
- Use bi-section search to find the p such as

$$E(T_{S_k{}^p})(1+Coe.(T_{S_k{}^p}))$$

is minimum

SCS

QoS Guided Min-Min Heuristics

for all tasks t_i in meta-task M_v (in an arbitrary order)
    for all hosts m_j (in a fixed arbitrary order)
        $CT_{ij} = ET_{ij} + d_j$
do until all tasks with high QoS request in M_v are mapped
    for each task with high QoS in M_v, find a host in the QoS qualified host set- that obtains the earliest completion time
    find the task t_k with the minimum earliest completion time
    assign task t_k to the host m_l that gives it the earliest completion time
    delete task t_k from M_v
    update d_l
    update CT_{il} for all i
end do
do until all tasks with low QoS request in M_v are mapped
    for each task in M_v find the earliest completion time and the corresponding host
    find the task t_k with the minimum earliest completion time
    assign task t_k to the host m_l that gives it the earliest completion time
    delete task t_k from M_v
    update d_l
    update CT_{il} for all i
end do

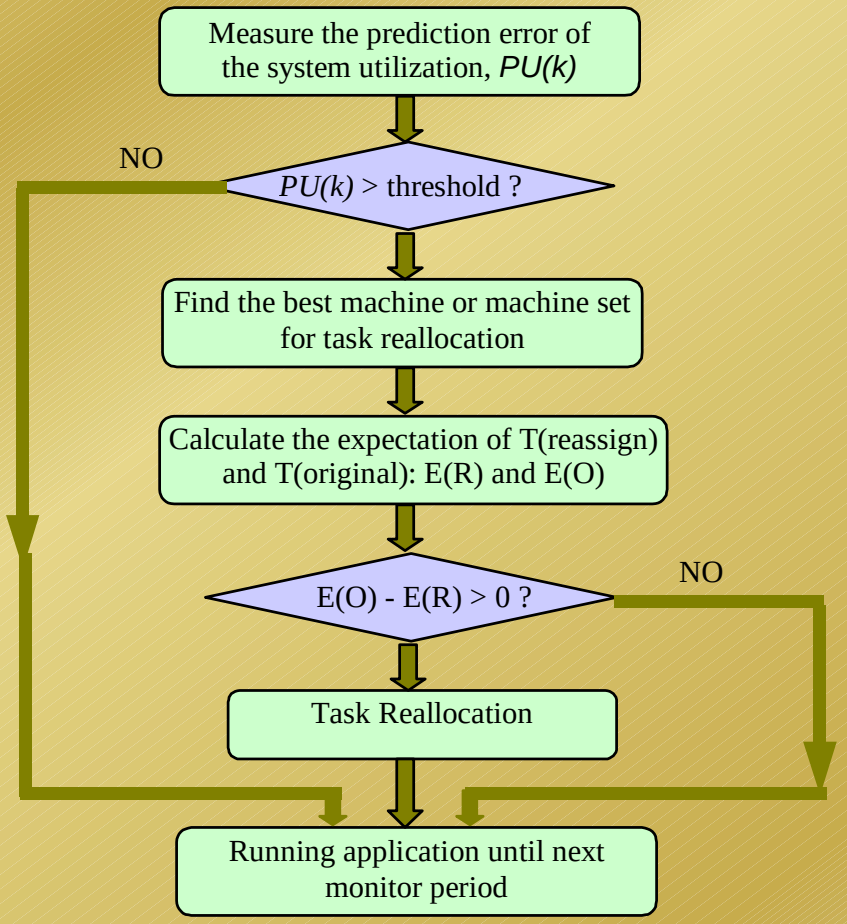
SCS

Data-Conscious Scheduling Heuristics

For a group of tasks T_i
    Get MCT_0 based on data replica placement.
For each task in the metatask but not run yet
    If subtask t_{ij} needs data d_n
        For all S_m in *Sites*
            Compute MCT_m based on *DDT* from host 1 to *m*.
            Get minimum MCT_{min}
        Endfor
        If $MCT_{min} < MCT_0$
            Copy dataj from host 1 to host *r*
            $MCT_0 = MCT_{min}$
        Else
            Do not copy
Endfor

SCS

Rescheduling Algorithm



SCS

GHS Design: System Architecture

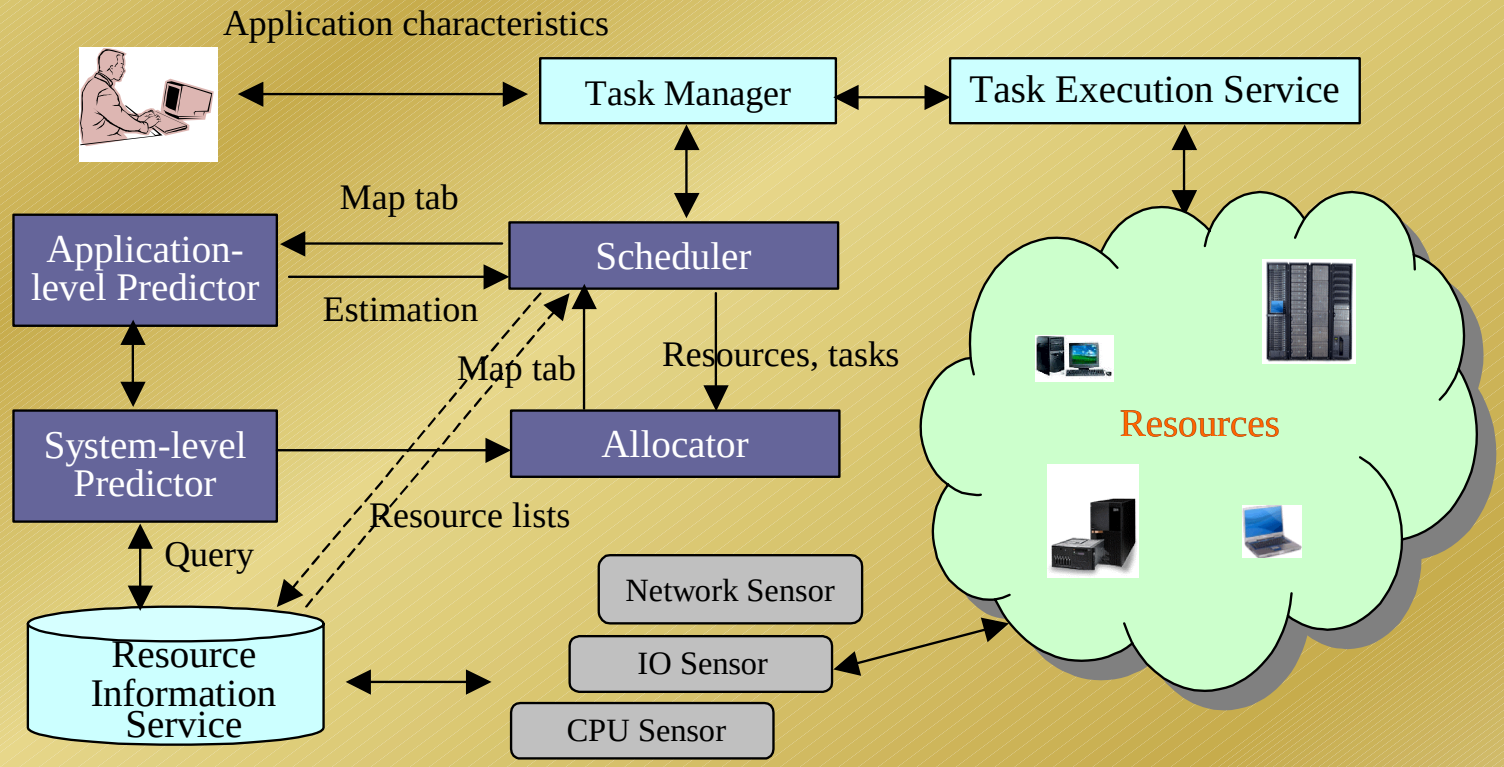


Figure. 1. A framework of GHS task scheduling system

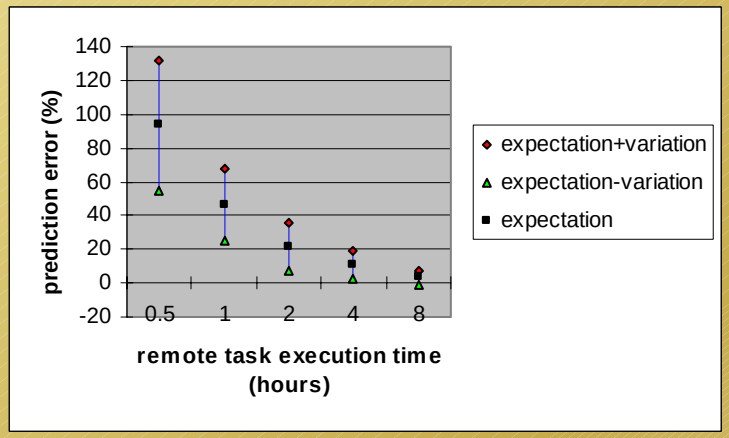
X. Sun, Feb. 2004

SCS

Experimental Testing

Application-level Prediction

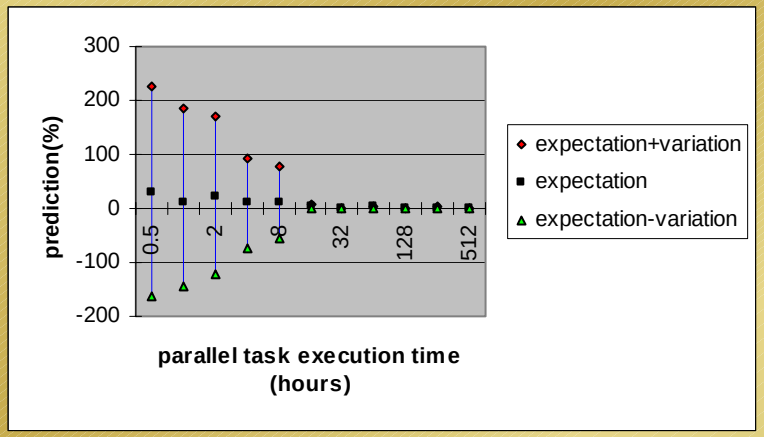
$$\left| \frac{\Pr ediction_{period} - Measuremen\ t}{Measuremen\ t} \right|$$



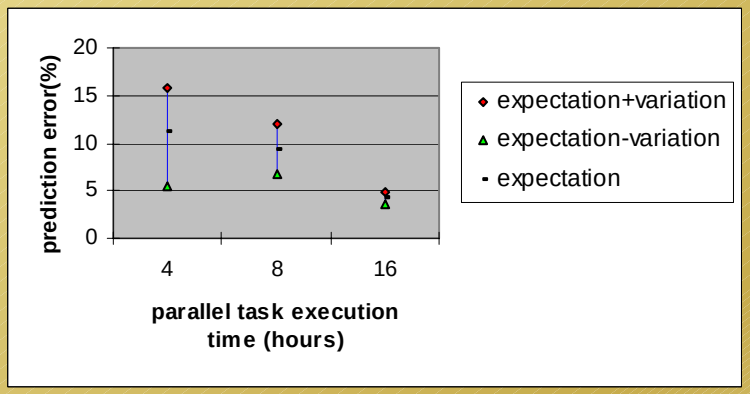
Remote task completion time on single machine

X. Sun, Feb. 2004

SCS



Prediction of parallel task completion time

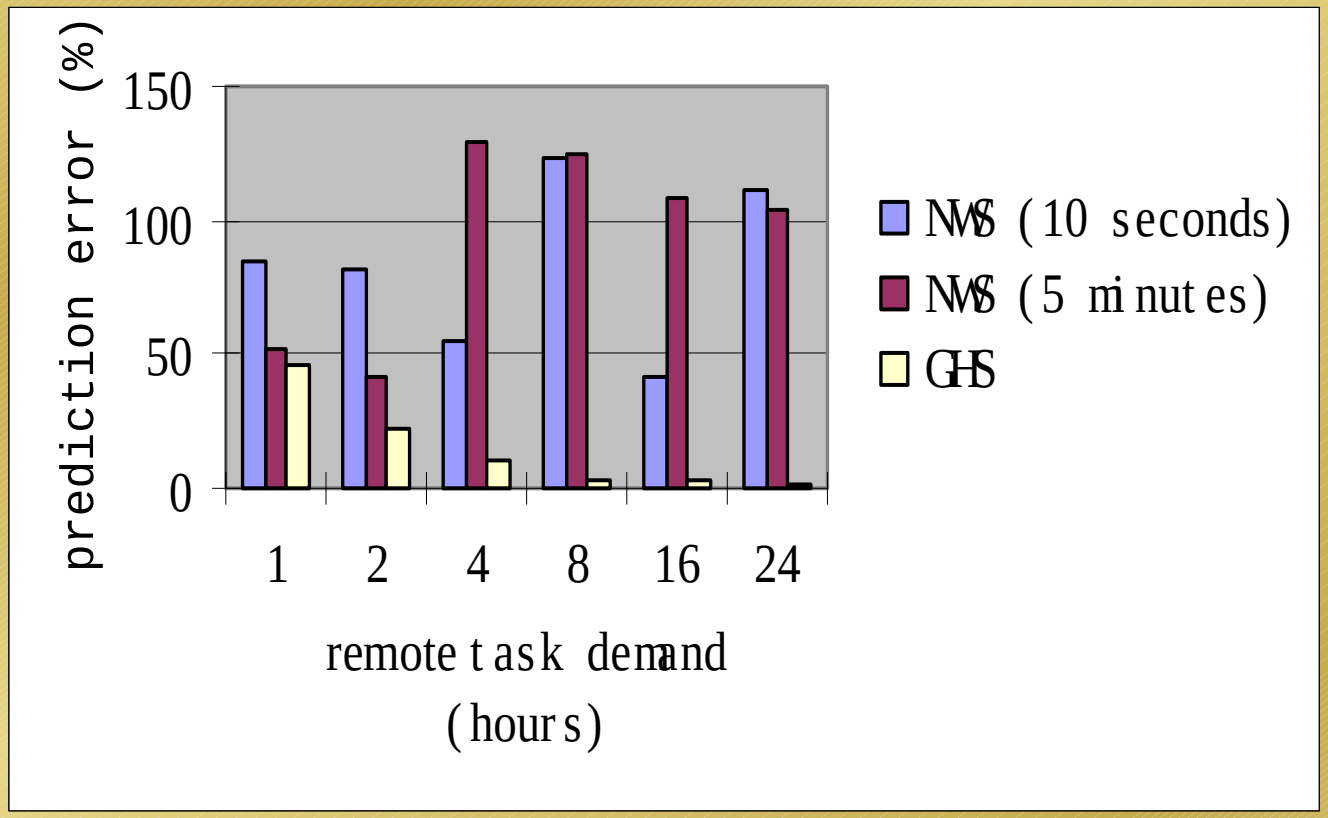


Prediction of a multi-processor with local scheduler

X. Sun, Feb. 2004

SCS

Comparison with NWS

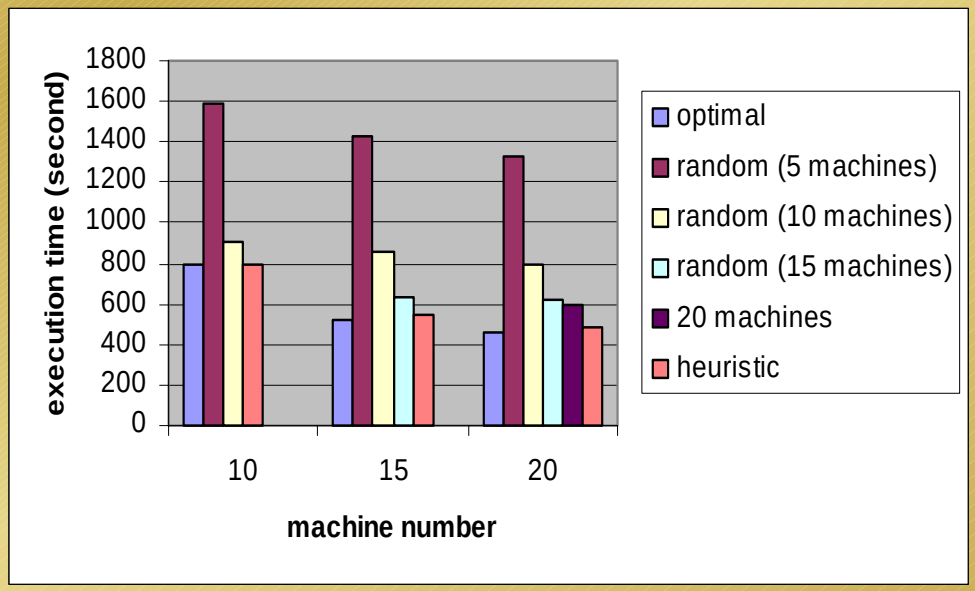


Mean of the prediction error of NWS and GHS

X. Sun, Feb. 2004

SCS

Performance Gain with Scheduling



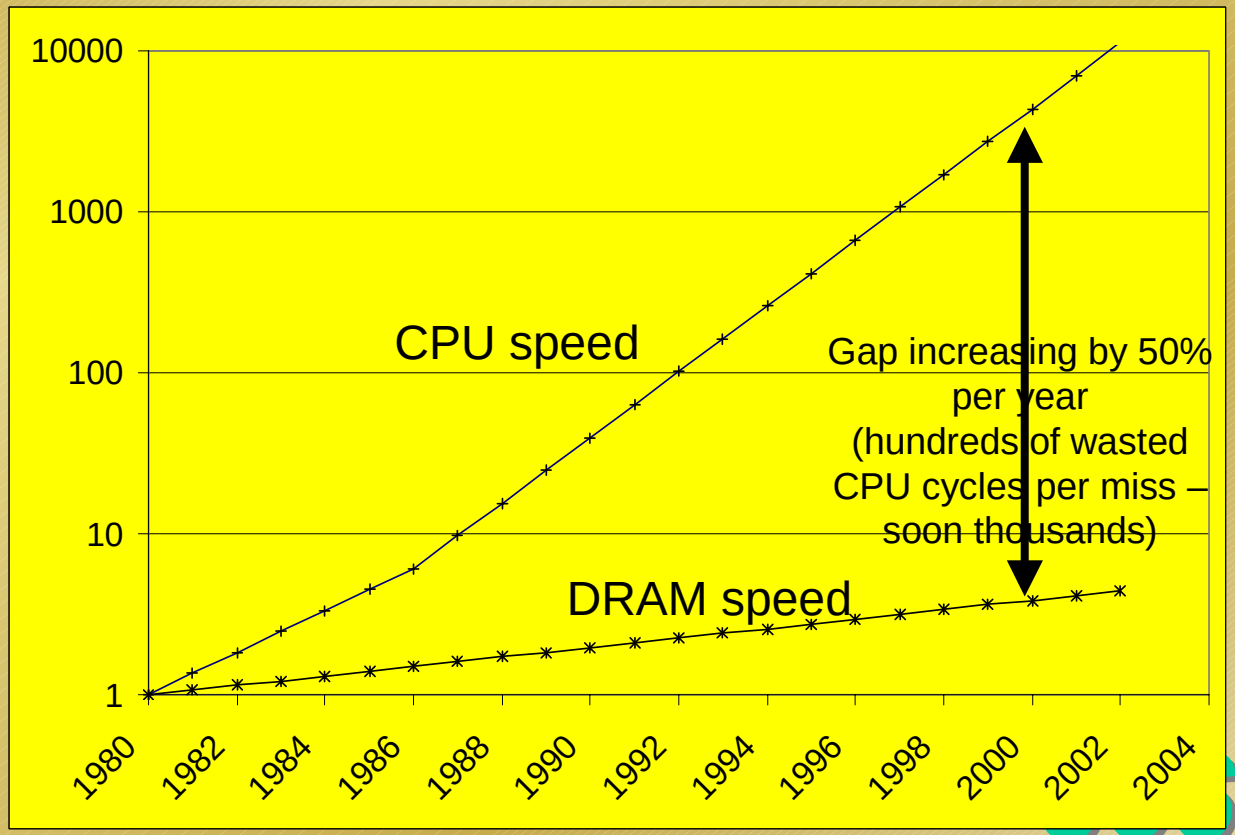
Execution time with different scheduling strategies

SCS

Data Access Optimization

- Data access is the bottleneck of high performance computing

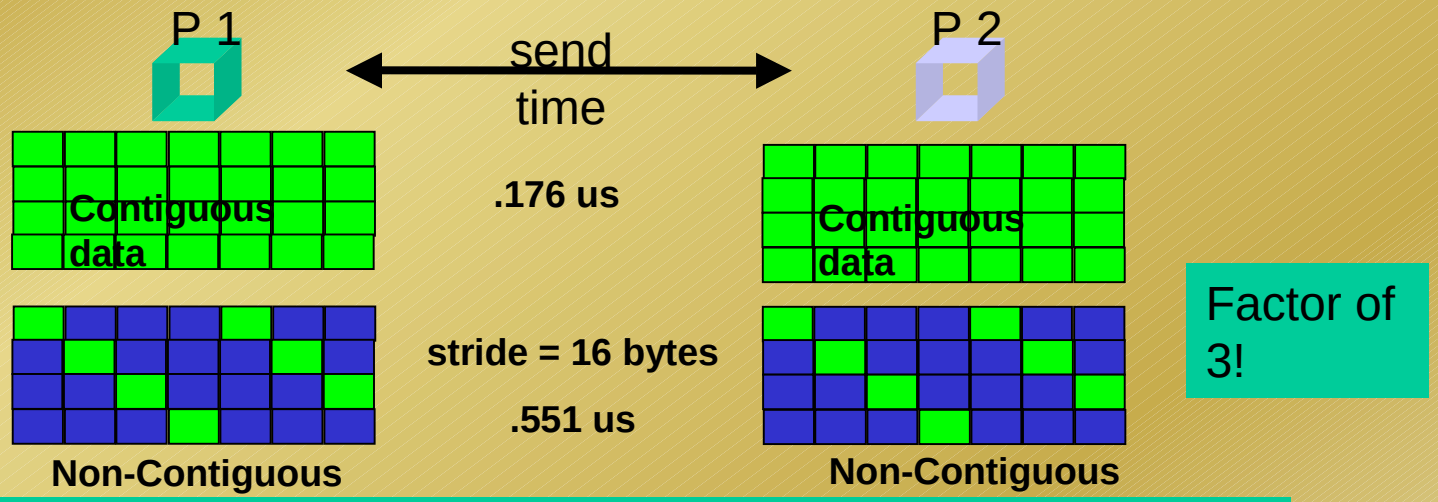
Relative Speeds of CPU verse DRAM



Much worse for I/O bottleneck (millions of wasted CPU cycles per miss)

Solution Approach

- Memory access pattern significant impact on comm.
- Impact differently on different machines
- The memory LogP model
- Optimization methods
- Application on MPI applications

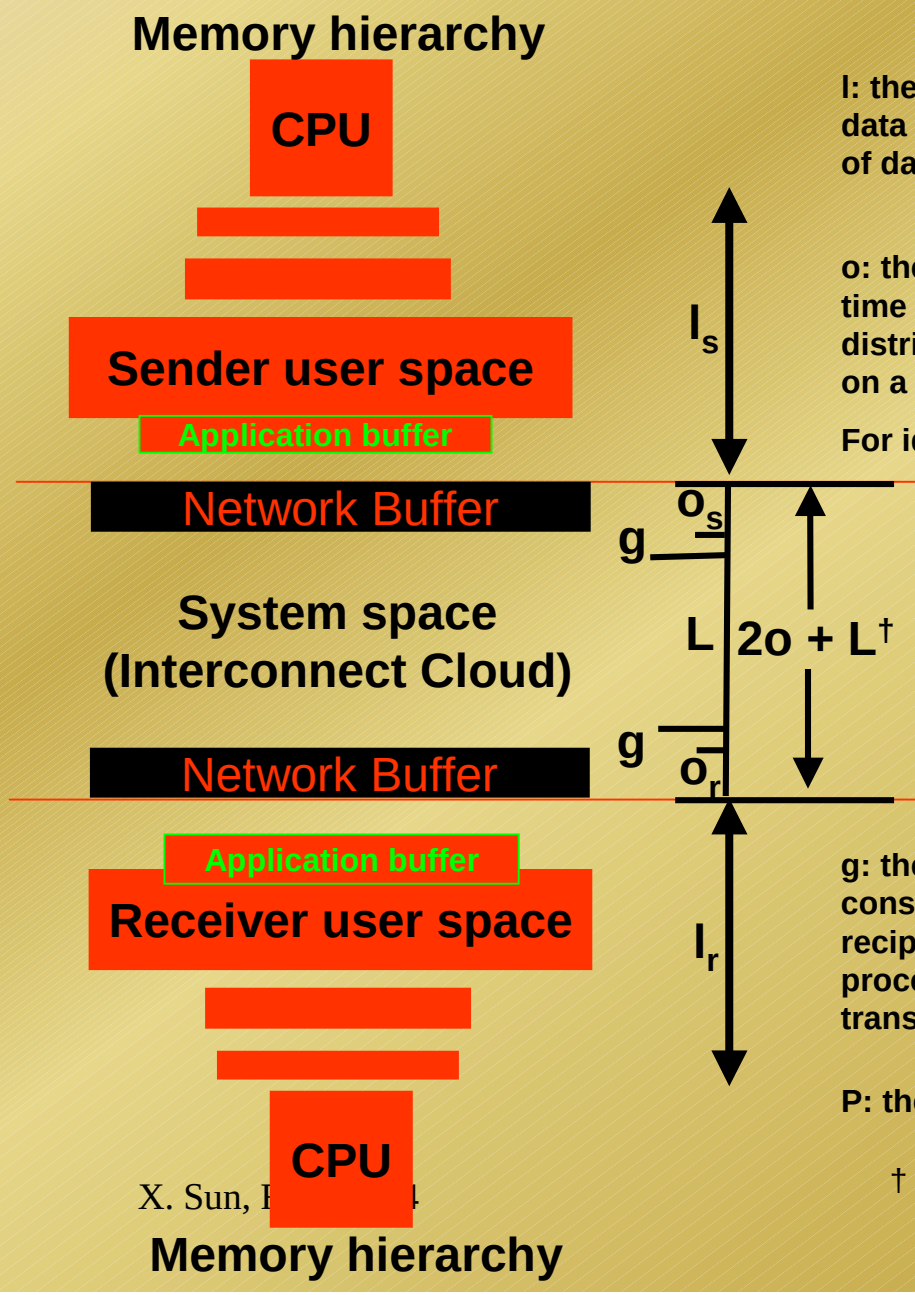


Factor of 3!

Memory communication cost can be significant for real applications.

SCS

The memory logP model



l: the additional latency in transfer non-contiguous data of data size (s) and distribution (d) for a given implementation of data transfer on a given system, l=f(s,d)

o: the data transfer overhead, defined as the length of time to transfer contiguous data of data size (s) and distribution (d) for a given implementation of data transfer on a given system, l=f(s,d)

For ideal case, o remains constant with size.

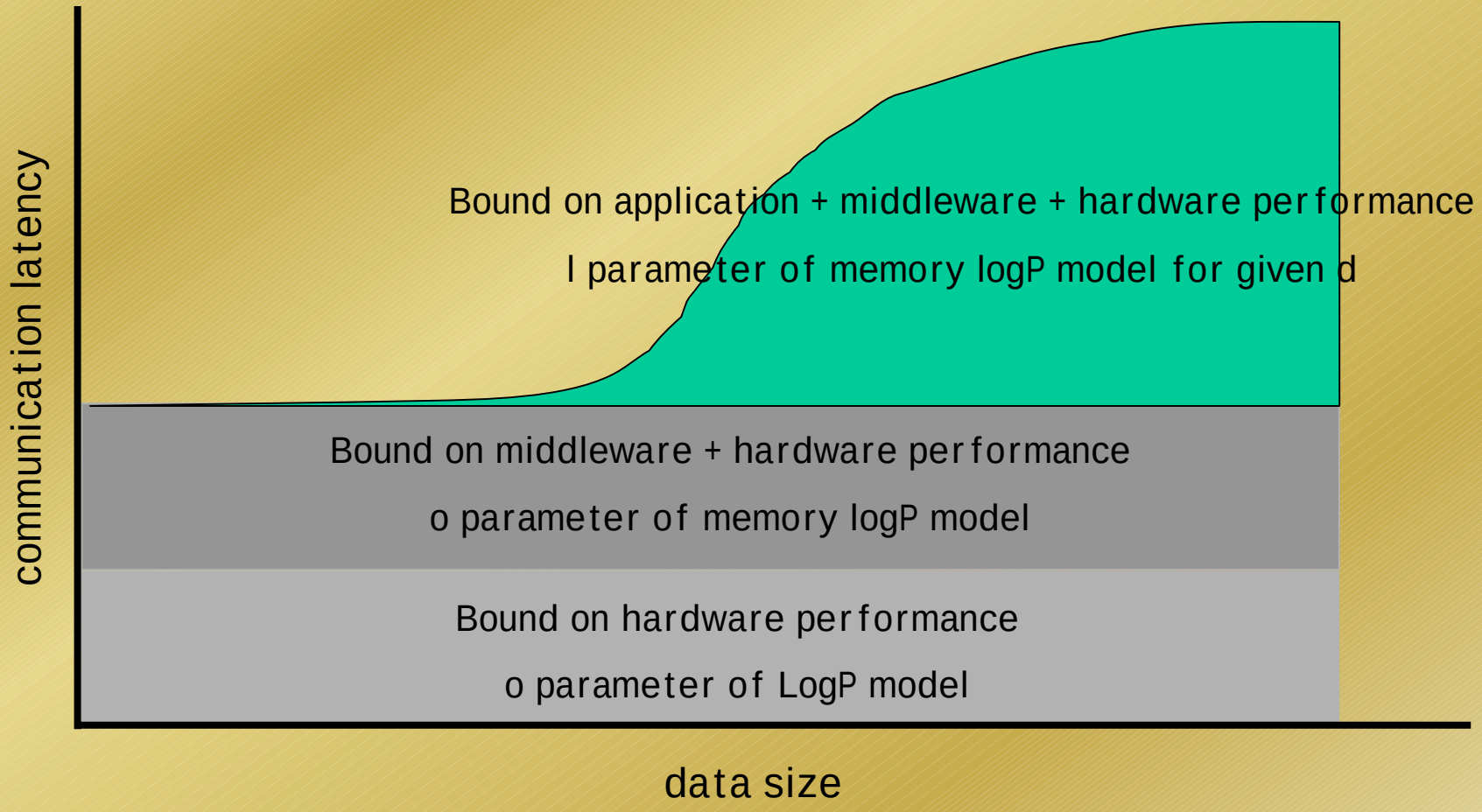
Total Comm Cost = 2(o+l) + L

g: the gap, defined as the minimum time interval between consecutive message receptions at a processor. The reciprocal of g corresponds to the available per-processor bandwidth for a given implementation of data transfer on a given system.

P: the number of processor/memory modules

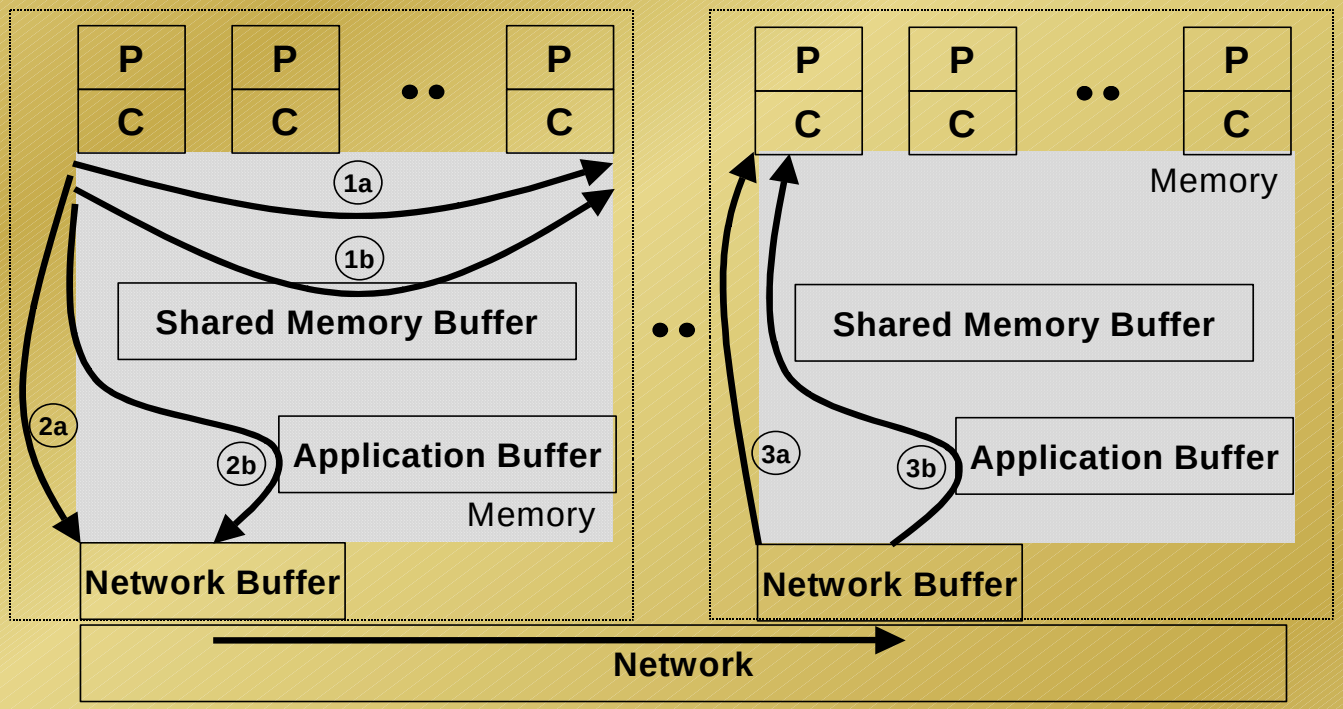
† **Assume g = o, common simplification of LogP model**

Performance bounds using memory-logP



SCS

Case Study: *Matrix Transpose on a SGI O2000 using MPI middleware for communication*

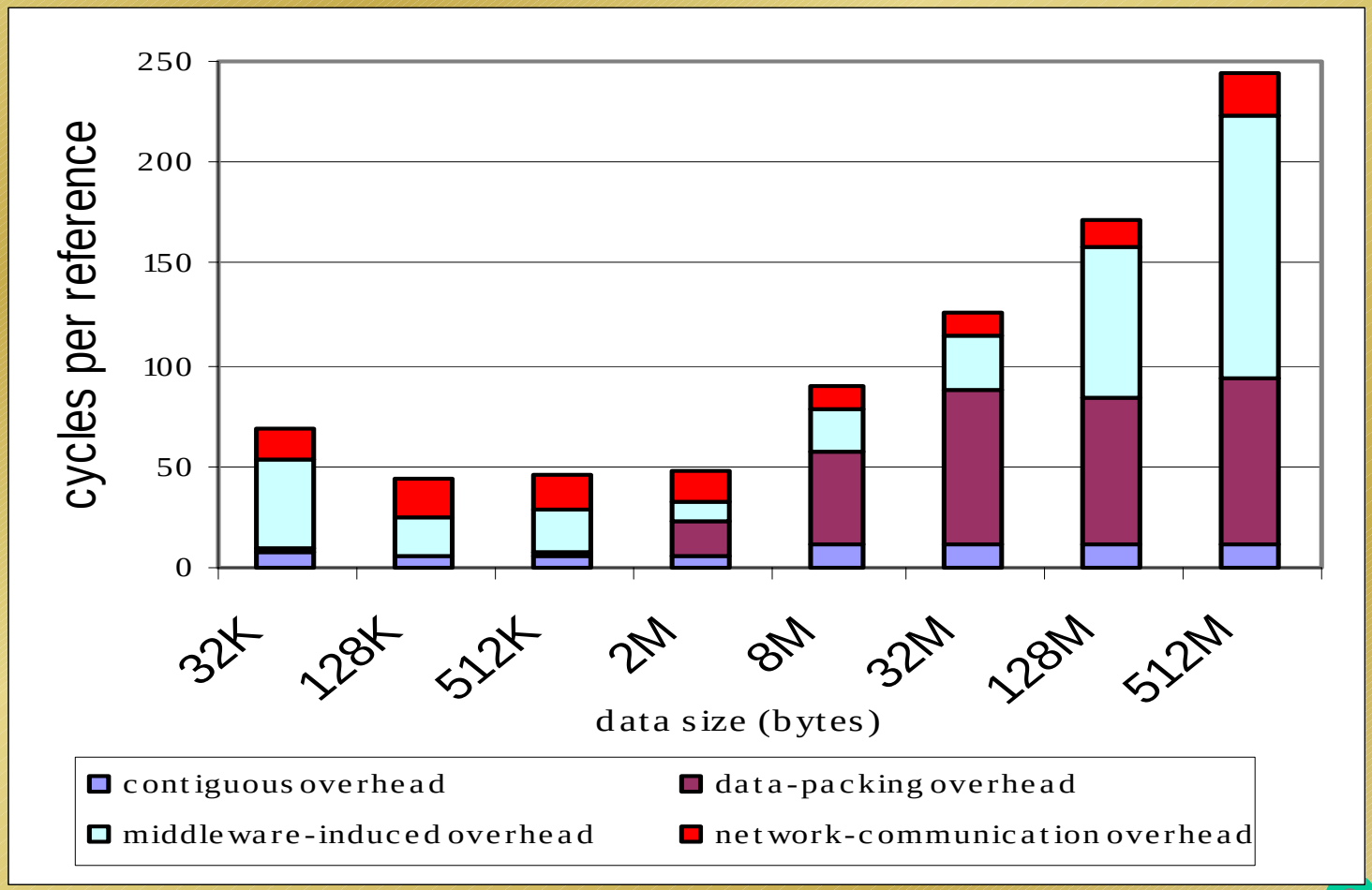


Comm. critical paths: Non-Cont. data need go through the application buffer first

X. Sun, Feb. 2004

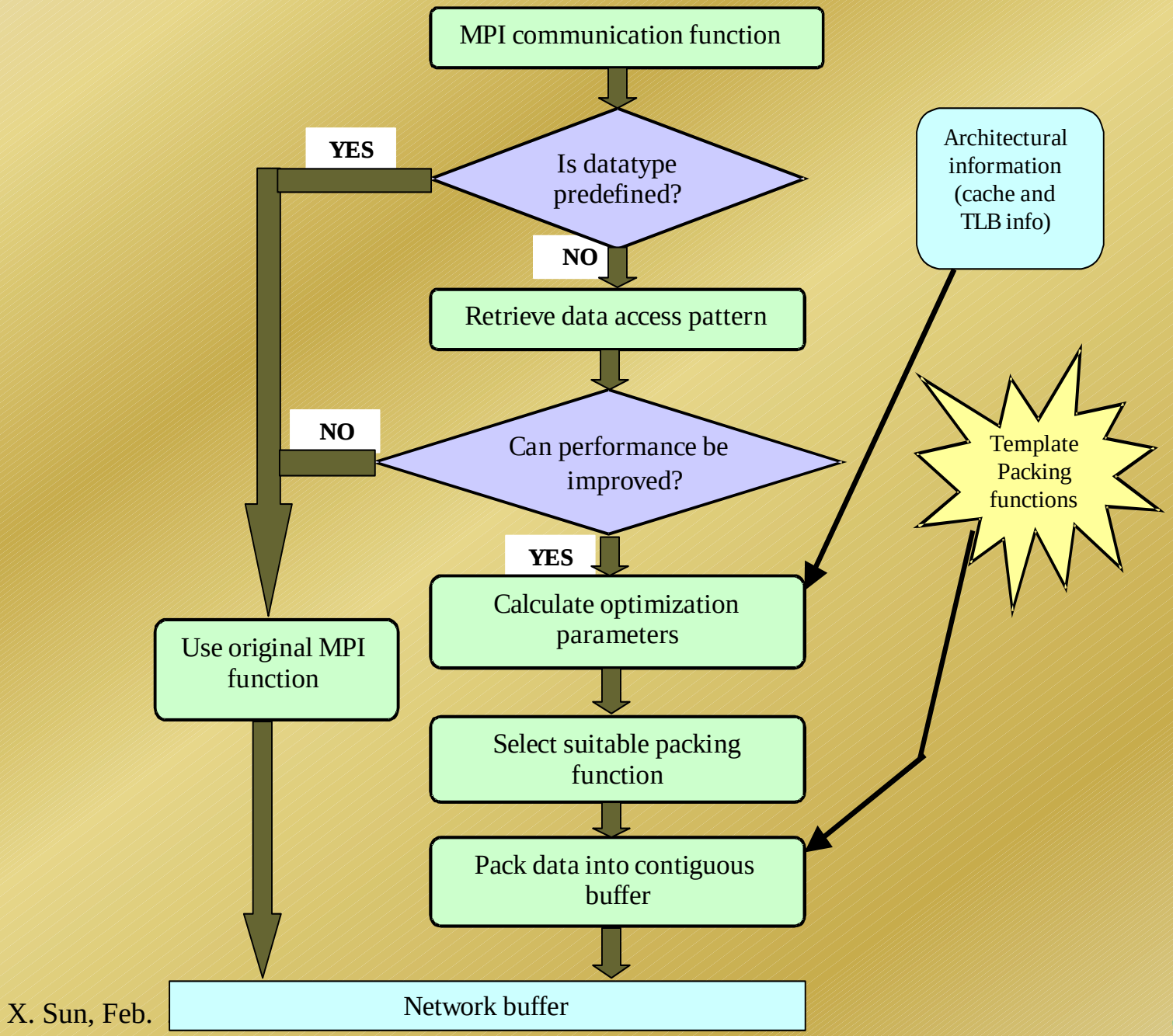
SCS

Quantifying Communication Cost for Matrix Transpose



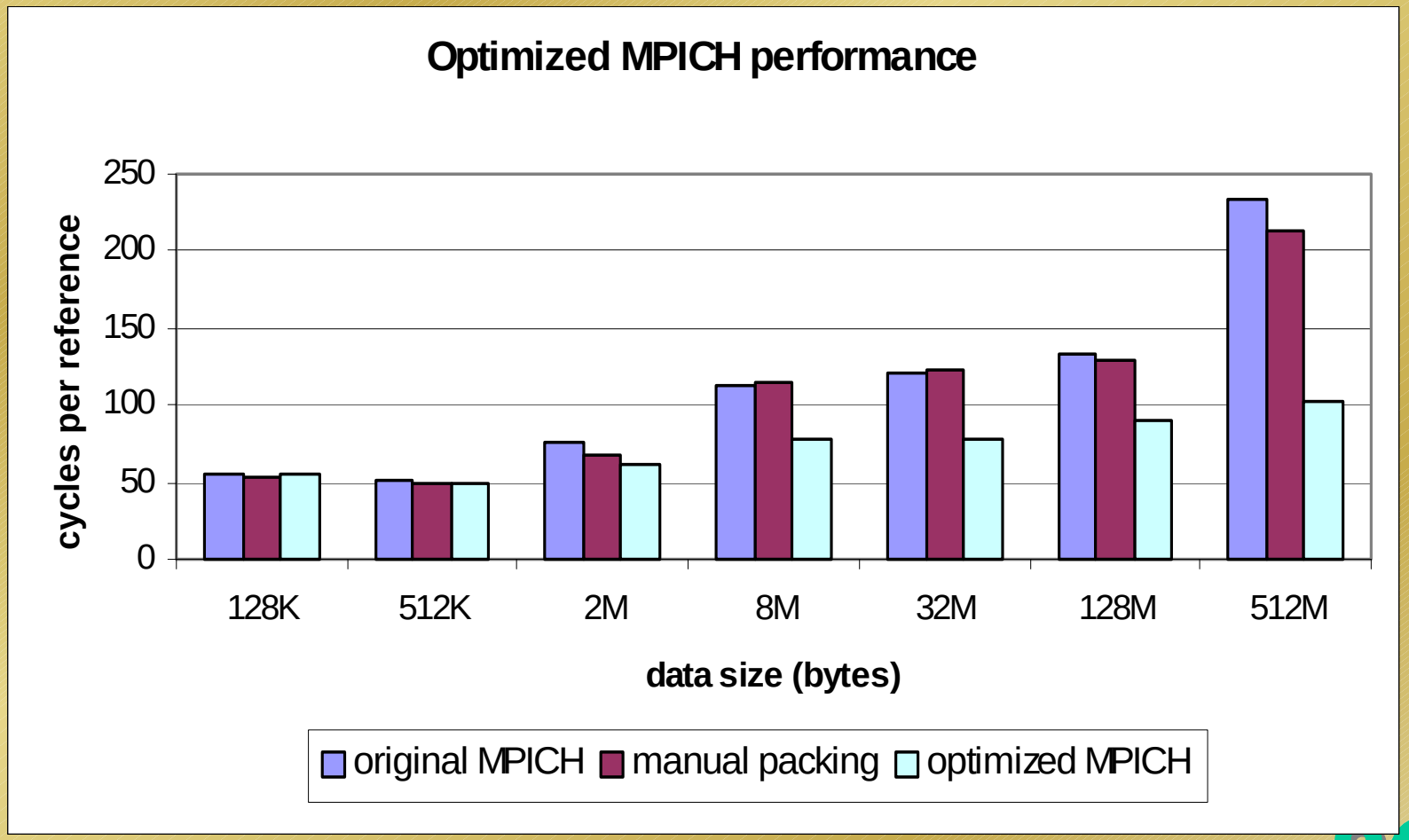
X. Sun, Feb. 2004

SCS


MPI communication function
Is datatype predefined?
YES
NO
Retrieve data access pattern
Can performance be improved?
NO
YES
Use original MPI function
Calculate optimization parameters
Select suitable packing function
Pack data into contiguous buffer
Network buffer
Architectural information (cache and TLB info)
Template Packing functions

SCS

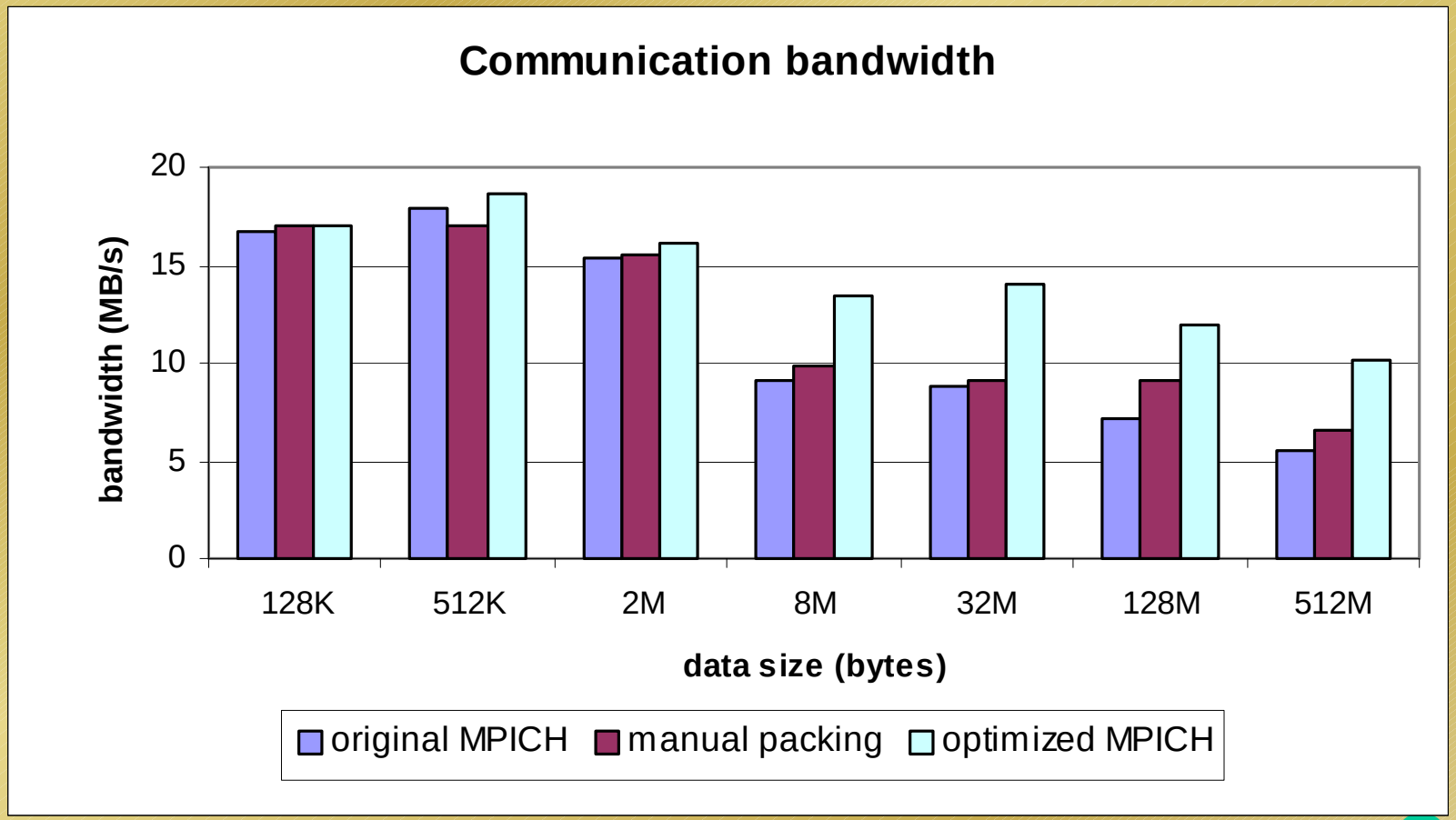
Improved performance of MPI derived datatypes



X. Sun, Feb. 2004

SCS

Bandwidth Improvement



SCS

The Cosmology Application

- Worked on a cosmology application called ENZO
  - SAMR (Structured Adaptive Mesh Refinement) application in astrophysics and cosmology
  - Developed by G. Bryan and M. Norman
- Three major tasks:
  - Conducted a detailed performance analysis
  - Developed a novel dynamic load balancing (DLB) scheme for ENZO on parallel systems
  - Extended the code to the Computational Grid

SCS

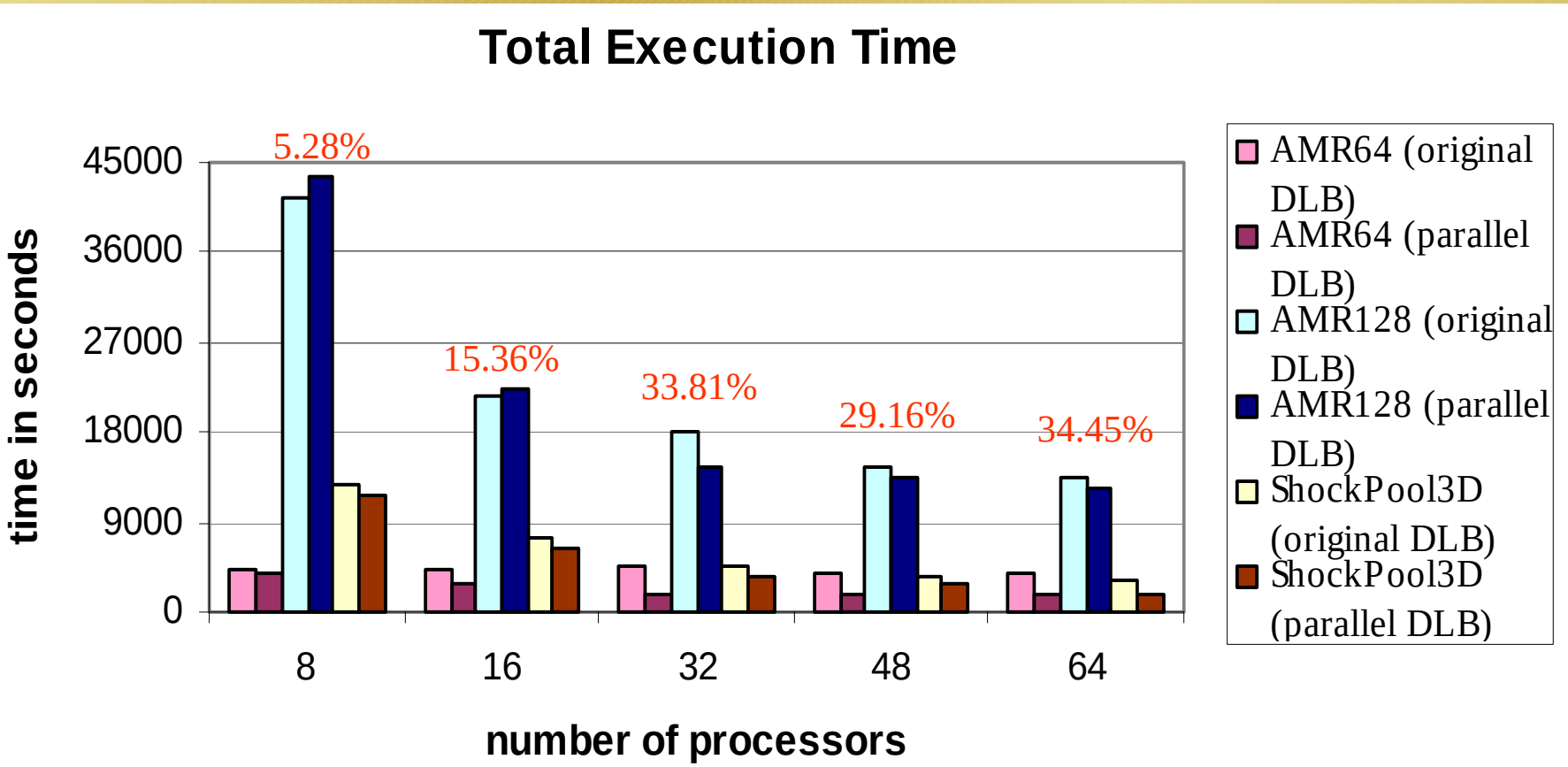
Task 1: Performance Analysis

- Overall characteristics
  - Manual instrumentation with FPMPI tool
- I/O performance
  - Use Pablo toolkit
- Adaptive characteristics
  - Coarse granularity
  - High magnitude of imbalance
  - Different patterns of imbalance
  - High frequency of refinements

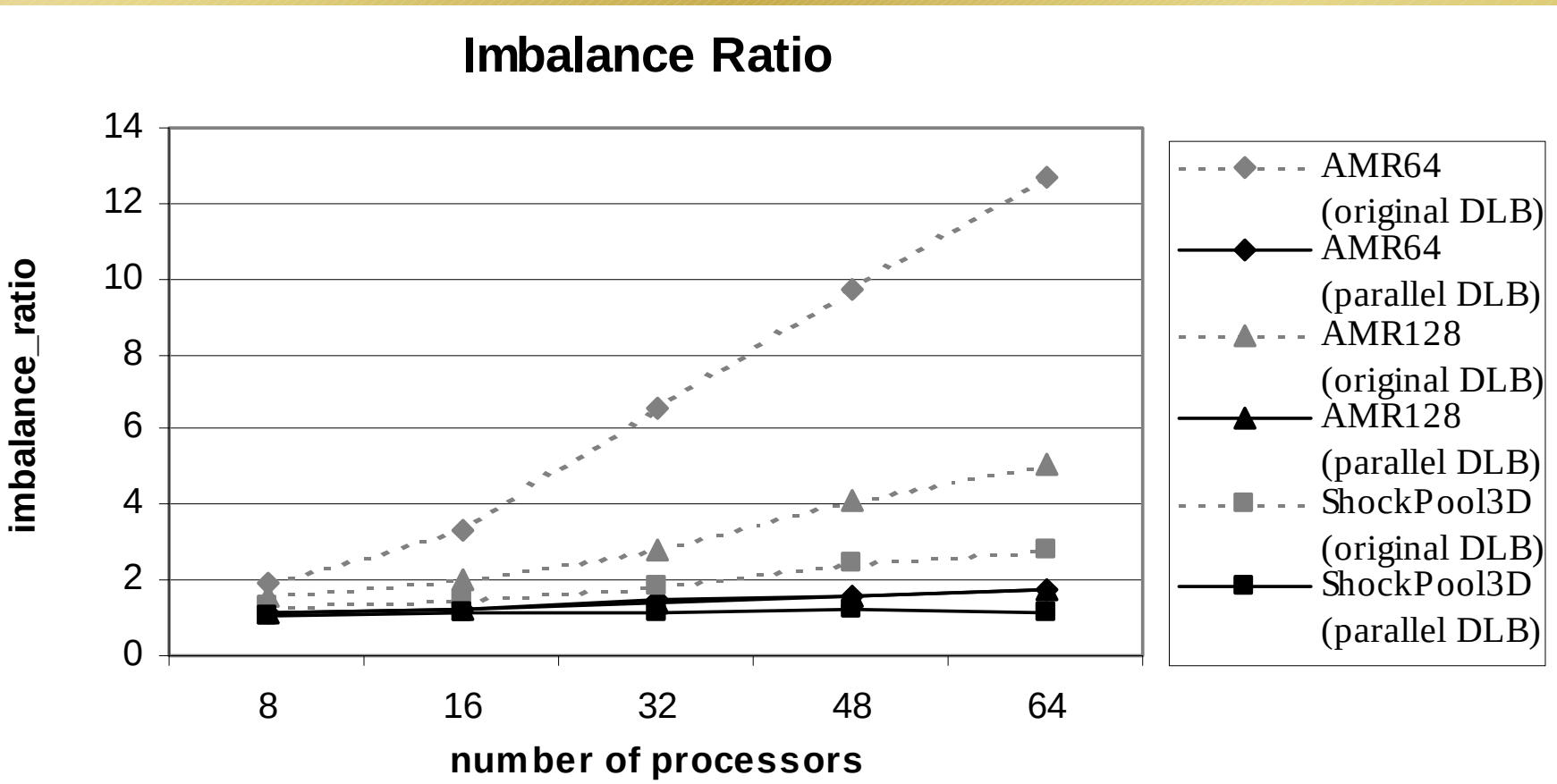
Task 2: Parallel DLB

- Design a DLB scheme for SAMR applications considering their adaptive characteristics
- Moving-grid phase
  - Directly move excess grids from overloaded proc. to underloaded proc.
  - Minimize grid movements by global information
  - Address the high frequency and different patterns
- Splitting-grid phase
  - Split the largest grid on the overloaded proc.
  - Address the coarse granularity

SCS


Total Execution Time
time in seconds
45000
36000
27000
18000
9000
0
5.28%
15.36%
33.81%
29.16%
34.45%
8
16
32
48
64
number of processors
AMR64 (original DLB)
AMR64 (parallel DLB)
AMR128 (original DLB)
AMR128 (parallel DLB)
ShockPool3D (original DLB)
ShockPool3D (parallel DLB)

SCS



$$imblanace\_ratio = \frac{MaxLoad}{AvgLoad}$$

X. Sun, Feb. 2004

SCS

Task 3: Extension to Grid

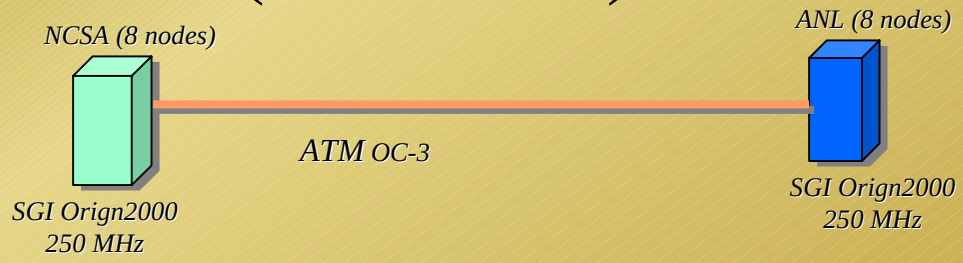
- Extend the improved version to the Grid
  - Alliance's Grid
- One major issue: a new DLB scheme for distributed environment
  - Heterogeneity of processors
  - Heterogeneity of networks
  - Dynamic features of networks
  - Adaptive features of applications

SCS

Experiments

- DS1: LAN-connected

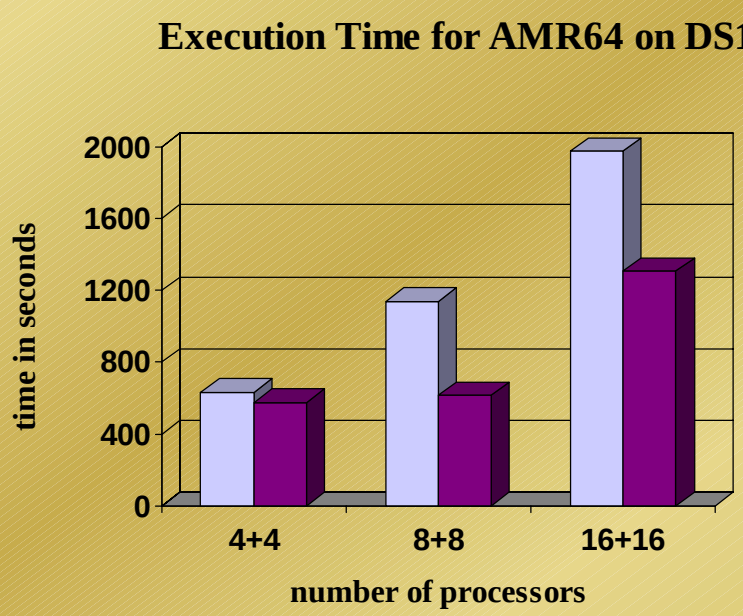
- DS2: WAN-connected (two locations)

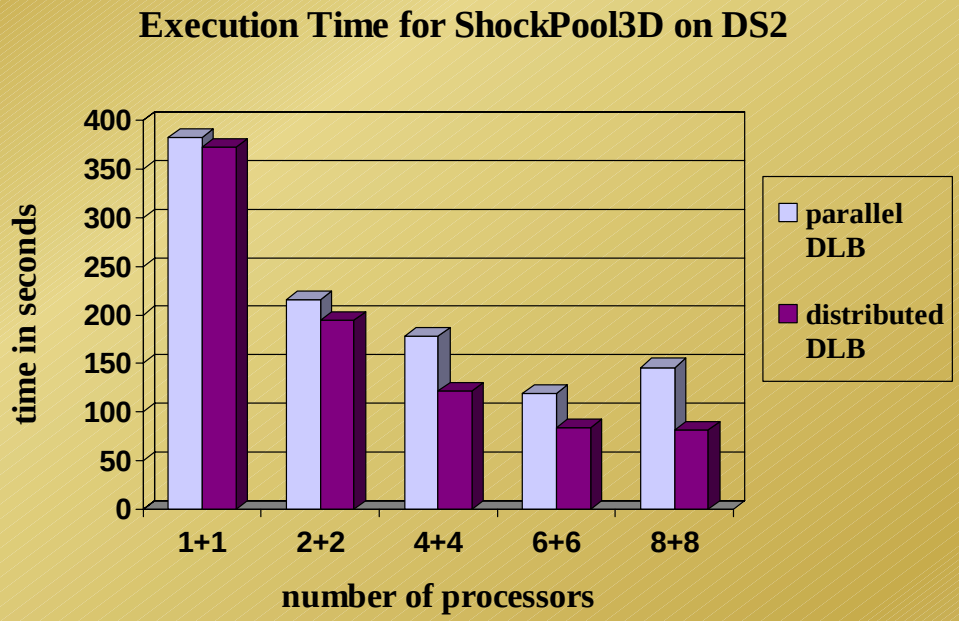


- DS3: WAN-connected (two countries)

X. Sun, Feb. 2004

SCS

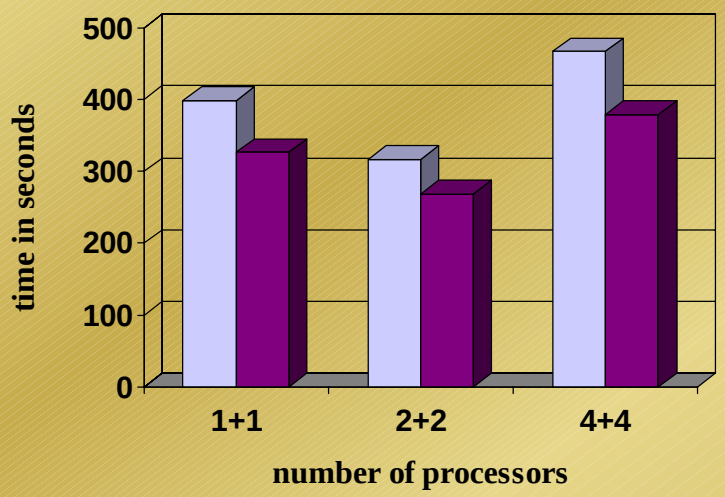




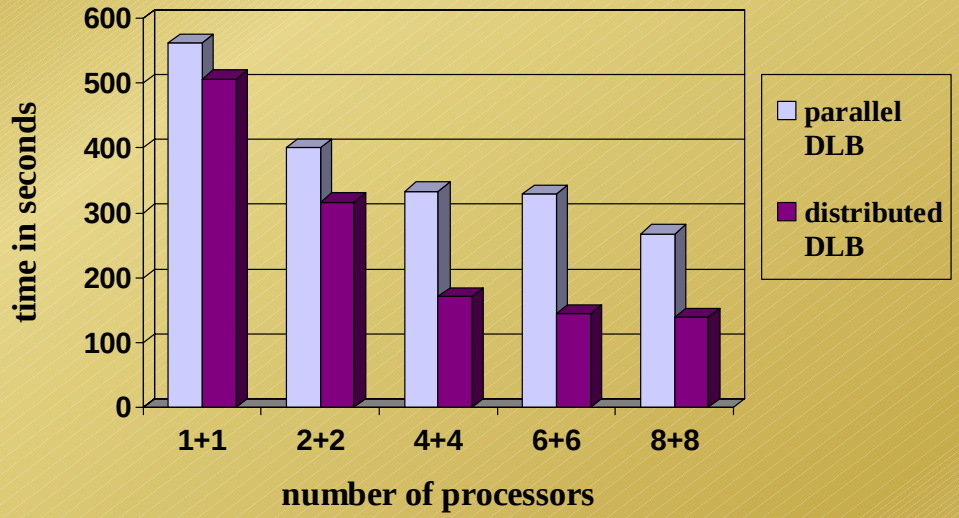
☞ The relative improvement ranges from 2.6% to 45.9%

SCS

Execution Time for AMR64 on DS3



Execution Time for ShockPool3D on DS3



☞ The relative improvement ranges from 10.0% to 56.1%

X. Sun, Feb. 2004

SCS

Conclusion

- Memory Model and Algorithms: Improve high performance computing
- Mobility plus Dynamic Scheduling: improve performance, reliability, availability, QoS, and trustiness of distributed computing
- Application-level Performance Optimization
- Fermi Applications: Performance analysis and enhancement, implementation on Grid, such as the TeraGrid

SCS